International Journal of Computational and Experimental Science and ENgineering (IJCESEN)

Vol. 11-No.4 (2025) pp. 4758-4765
http://www.ijcesen.com

ISSN: 2149-9144

Copyright © IJCESEN

Research Article

Architectural Decision Frameworks for Large-Scale Salesforce Implementations: Balancing Configuration and Custom Development

Santhosh Kumar Balachandran*

University Of Texas, Austin
* **Corresponding Author Email:** santhoshataus@gmail.com- **ORCID:** 0009-0000-9620-2114

Article Info:

DOI: 10.22399/ijcesen.5365
Received : 01 March 2025
Revised : 29 March 2025
Accepted : 30 March 2025

Keywords

Salesforce Architecture,
Well-Architected Framework,
Enterprise CRM,
Apex,
Flow,
Custom Development

Abstract:

Large-scale Salesforce programs continually face the decision of whether to meet a business requirement through declarative configuration or programmatic customization. Configuration accelerates delivery, lowers cost, and stays aligned with platform upgrades, while custom development with Apex, Lightning Web Components, and external services offers control over differentiated processes and complex integration. This paper presents a decision framework that helps enterprise architects, platform owners, and governance boards navigate this trade-off consistently across business units. The framework is grounded in the Salesforce Well-Architected Framework (Trusted, Easy, Adaptable) and the published Salesforce Architect Decision Guides, and is organized around five decision variables: business process uniqueness, time-to-value, integration complexity, regulatory and security requirements, and long-term maintainability. We argue that a configuration-first approach with selectively governed customization, supported by an explicit architecture pipeline and periodic technical-debt reviews, is the most sustainable operating model for enterprise Salesforce.

1. Introduction

Today, Salesforce has evolved into an enterprise platform that drives all customer engagement, service operations, partner channels, regulated workflows, and enterprise reporting. Its cadence of delivering many business processes declaratively, via custom objects, with three releases per year and a metadata-driven architecture, means it delivers many business processes. It also has a wide level of customization through Apex, Lightning Web Components, Platform Events, External Services, and integration with MuleSoft, Heroku, and Data Cloud on the same platform. This brings a 2-track delivery style in which the same thing can often be addressed via clicks, code, or both.

In enterprise-scale programs, solutions are rarely straightforward or limited to a simple yes-or-no approach. Multinational deployments involve several complexities, including compliance with data residency regulations, management of federated identities, integration of legacy systems, support for mergers and acquisitions, coordination across multiple Industry Cloud verticals, and the administration of tens of thousands of users spanning diverse business functions. Here, configuration can speed up the initial release, but without proper control, it can lead to automation sprawl and conflicting validation logic. There might be something you need to fill in with "hacky" code, but you will introduce technical debt that could slow the development process and provide specialist skills that are less common in your organization. Both risks are backed by empirical evidence related to the technical debt and low-code platforms [1][2][3][4].

While the Salesforce platform's mature, enterprise-level architectural design research is still underrepresented. Vendor documentation is used to highlight a click-event-based story, while academic case studies are generally geared towards smaller installations. Salesforce is now making an effort to tackle this with a series of Architect Decision Guides to explain how to build Salesforce record-triggered automation, forms, data integration, and event-driven architecture [5] and with a release of the Well-Architected Framework, which outlines Salesforce architectural decisions as being either trusted, easy, or adaptable [4]. In particular, the latest iteration of the Record-Triggered Automation Decision Guide changed the basis for deciding between Flow and Apex from who is responsible

for building to the object's automation density across the entire application [6]. As of yet, these are authoritative but aren't currently defined in a single business-unit decision framework, driven by governance, and made available to business units by an enterprise architect.

The guidance presented herein validates this up to date into a framework. It covers 5 decision variables: process uniqueness, time-to-value, integration complexity, regulatory & security considerations, and maintainability, and incorporates them into an architecture pipeline in connection with the Well-Architected pillars. It's designed for enterprise architects, platform owners, CIOs, and enterprise architecture review boards managing large Salesforce estates.

2. Related Work

Cloud CRM adoption, low-code platforms, technical debt, and DevOps are all topics covered by the research, which is relevant to enterprise Salesforce architecture. Research on implementing cloud-based CRM systems demonstrates increased agility and responsiveness in customer handling, yet still implementing the basic features of the system as is [7, 8]. The vast majority of developer challenges arise after declarative features and extend to extensions, integrations, and complex UI logic [3] [9]. Research on technical debt in SaaS shows that code debt is not the only way to accrue debt; it also arises from shifts in the config, interacting automations, and security model, creating technical debt through drift [1][10]. In platforms with custom extensions, DevOps research focuses on the frequency of changes (version control, release pipelines, automated testing) and on the speed and safety of change [11].

This literature is further enriched with Salesforce-specific guidance. The Well-Architected Framework introduces a three-part approach, referred to as the lens (Trusted, Easy, and Adaptable), along with corresponding patterns and anti-patterns for each aspect [4]. The Architect Decision Guides provide prescriptive trade-off matrices with platform decisions that are most contested [5, 6]. The Enterprise Agentic Architecture guidance discusses the design of systems composed of multiple agents on Agentforce, with consideration of governance and inter-org orchestration [12]. The Data 360 Architecture document defines Data Cloud as the enterprise data layer with native support for Snowflake, Databricks, BigQuery, and Redshift [13]. So, the way "configuration-versus-custom" is defined in 2026 has expanded; it's no longer confined to standard functionality versus the configuration, but also includes Flow, Apex & LWC development, AppExchange packages, MuleSoft, and other external services.

3. Architectural Decision Variables

In short, the 5 variables mentioned above are always at play across various initiatives within a company's Salesforce. They all support one or more of the Well-Architected Framework pillars: Trusted security, compliance, and reliability; Easy, intentional design, automation, and an engaging user experience; and Adaptable resilience and composability. For highly specialized processes, decision-making is rarely straightforward or absolute. A unique set of requirements combined with strict time-to-value expectations often leads to decisions that differ significantly from those made for conventional processes. The envisioned approach represents only one segment of a broader continuum that includes standard functionality, declarative configuration, Apex and LWC development, AppExchange/ISV solutions, and externalized logic through platforms such as MuleSoft, Heroku, or External Services.

3.1 Business Process Uniqueness

Process uniqueness is the primary signal for whether to configure or to build. Most enterprises run a mix of commodity, critical, and differentiating processes. Commodity processes, lead capture, standard approvals, and common case management are well served by configuration and should follow the platform's defaults to maximize upgrade compatibility. Processes that protect revenue, enhance the customer experience, or preserve pricing flexibility may justify selective custom development or an OmniStudio implementation when delivered on an Industry Cloud. Critical processes that carry operational or contractual risk usually resolve best as a hybrid: a configured backbone extended with targeted Apex or LWC where the platform's defaults fall short. The trap to avoid is custom-building a clone of a legacy process just to preserve familiarity. That path imports old constraints into the new platform and squanders the productivity gains the migration was supposed to deliver in the first place.

3.2 Time-to-Value Pressure

When this type of time-to-value pressure is felt, it is more helpful to adopt a configuration-first approach. In a couple of weeks, you'll see the visible end result, including regular objects, Flow-enabled dashboards, and permission sets, to enable

an iterative co-design process with business stakeholders. Customizing too much in the initial release will negatively affect the design, development, testing, and release schedule, and delay stakeholder validation compared to the intended schedule. The opportunity in this type of project is for the architect to distinguish between what is necessary and what is merely important, and then design and deliver only the minimum functionality: the functionality that is really needed is built into the platform itself, and the rest is added later through extensions. In addition to time-to-value, the organization's change capacity is required: change in users does not occur any faster than they learn what the new (platform) systems can and should do, and are able to use them.

3.3 Integration Complexity

For enterprise products, Salesforce is likely to be a component of a CRM. It can typically be integrated with a wide range of commercial applications, including ERP systems, Bill Processing Systems, Identity Providers, Data Warehouses, E-commerce, Support, and Partner Applications. Simple integrations with files, external services, and/or platform events or Change Data Capture (CDC). An integration platform is where complex orchestration, exception handling, and reconciliation live. Here, MuleSoft is Salesforce's first-party solution, taking the pressure off Apex triggers, custom objects, and other components where orchestration is the sender's responsibility, whether or not it was integrated or orchestrated by MuleSoft. If the goal is not orchestrating the process but rather data unification, Data Cloud (Data 360) provides a federated model that integrates data across Snowflake, Databricks, BigQuery, and Redshift without storing any extra data. The premise is to keep the Salesforce handle of what's happening most customer closer to the customer record, and move what can be thought of as "event transformation," "batch reconciling," and "orchestration of events across various systems" into another layer of integration. Being aware of them as early as possible is important so that the core org is not customized too early.

3.4 Regulatory and Security Requirements

Whereas the requirements from regulated industries such as banks, healthcare, telecommunications, insurance, and government set out requirements upfront, such as audit and access control, retention, and data residency. This is mostly addressed natively with profiles, permission sets, sharing rules, field-level security, audit trail, Shield Platform Encryption, environment management, and Env Monitoring. Hyperforce takes that one step further by promising regional deployment, with its data residing in 17 countries and 25 regions [14]. It is justifiable to custom-develop when the need exceeds what is available natively, bespoke consent management, jurisdictional data-processing rules, custom archival, advanced fraud detection, integration with an enterprise SIEM, etc. Custom code is added where the native control architecture 'wins', that is, does not duplicate it: secure development lifecycle, code review, static analysis, secret management, and least-privilege design, used for each custom component.

3.5 Long-Term Maintainability

At design time, the variable that is underweighted typically is “maintainability”, and it is dominant over the platform's lifetime. Also, it is easier to support a configuration-led solution, as administrators can extend it without relying on developers, and native features will be supported throughout releases. However, customization is more demanding and requires expertise in Apex, LWC, and integration, which are less available or more expensive than administrative skills, and it carries delivery risk due to limited expertise. Debts can be collected in all the predictable ways used when writing custom code, including duplicated logic, unused fields, generic SOQL, hardcoded references, and when undocumented automations are used, more debt comes in. Limits (CPU time, SOQL queries, DML rows, heap size, callouts, async job counts) are hard constraints; they can only be identified at design time, particularly when there is high automation density on an object [6].

There are two practices that have a profound influence on maintainability. One direction this is going is in replacing the "happy soup" of metadata with modular, versioned, source-controlled artifacts with Salesforce DX, unlocked packages, and second-generation managed packages. This is a component of the DevOps maturity ladder that is being implemented, starting with the system of truth (source control), moving through pipelines (DevOps Center or partner tools such as Copado, Gearset, Flosum), and finally to automated regression on a representative dataset and static analysis of Apex. These practices will apply even to a declarative or coded change being shipped. The second method is periodic architecture review, detailed in Section IV, which ensures that there isn't a time when the org ends up with changes that are really just swapping out native functionality. It must be monitored, along with delivery velocity, incidents reported to it, time-to-enhance, regression

effort, and retirements of customizations in each release. It should be captured alongside all other first-class metrics for delivery velocity: incidents, time-to-enhance, regression effort, and the number of customizations retired in each release.

4. Architectural Decision-Making and Governance Pipeline

These 5 business variables are only available in a repeatable pipeline. The demand itself is recorded, as are its sponsor, value, and need. The analysis should explain how the requirement will be met when using the core functionality, default configuration, or when using Flow, when using a MuleSoft App (managed package from AppExchange), an Apex or LWC component, or by an external module not a part of MuleSoft (on MuleSoft, Heroku, or external services). If there are multiple options available, the corresponding Architect Decision Guide should be referenced; in cases where logic is triggered on records, the choice is not so much whether to use Flow or Apex, but how much logic (when reaching the target object). From then on, all subsequent requests for the same object follow the same logic (e.g., the decision and reasoning are captured by the architecture review board).Continuous delivery takes place via DevOps Center or other toolchain, within a sandbox, in sprints, with version control, automated testing, and a release pipeline. Partial Copy Sandbox is an architectural option commonly used in multi-team programs and temporary UAT environments, enabling teams to test system behavior, validate integrations, and collaboratively assess their overall impact. The problem of a single-organ (as opposed to a multi-organ) approach is accepted, but for global programs, guidance on publishing for the quadrants Unification, Coordination, Replication, and Diversification is the entry point [15]. Typically, a federated governance option is the right choice for multiple regions or business units. The shared vision/standards will be owned by the global Center of Excellence (CoE), with interpretation and regulatory demarcation (and business-unit priorities) owned by regional architecture review boards (ARBs). The separation is designed to ensure consistency across the program while still allowing flexibility for local teams, reducing the likelihood of bypassing established processes. However, this approach is effective only when supported by clearly defined decision-making guidelines; otherwise, it may result in extended negotiations and the rise of shadow IT practices. Additionally, periodic architectural reviews remain essential. It is also important to account for the platform's annual release cycle, which includes Spring, Summer, and Winter releases. What is considered a customization could be "out" in just 2 years, or what we consider a customization may become a native Salesforce Flow. At this stage, when the program has reached a significant level of maturity, each post-release cycle becomes a process of re-evaluation. After deployment, the team identifies any gaps or newly introduced native features that were previously unavailable and begins planning the retirement of functionalities that are no longer necessary, including custom code, complex automations, and installed AppExchange packages. The number of customizations built out per release should also be included in the list of portfolio health components, along with the adoption, incident, time-to-enhance, and automation-failure rates. However, if it is not part of this ritual, the technical debt grows unwittingly, and the program eventually has to pay for it anyway.

5. Empirical Grounding of the Framework

The central tenet of the framework, "configuration is the default, customization is the exception, justifying that," is supported by empirical evidence from both cloud CRM adoption research and low-code, configuration management, and DevOps research. Typical CRM deployments are consistently found to be expedient, less expensive, and easier for users to adopt than other standard-capability-related deployments [7, 8]. The difficulties and unanswered questions reported by low-code researchers for each developer project are mostly related to customization, integration, and complex UI logic, further underscoring that custom development is out of the ordinary, not the norm [3], [9]. Along with code, configuration research cautions that "uncontrolled declarative work creates its own 'click' debt: Many people were warned of the automation sprawl and conflicting rules created by it, which is why the extension of governance into the clicks was just as well deserved. DevOps research correlates source control and automated testing with release pipelines to enable safer, faster change, thereby reinforcing the use of engineering principles in everything we make [11]. Further developments in Salesforce engineering were recently released for Hyperforce, Data Cloud governance, and Agentforce, confirming the findings above in the context of a 2026 target horizon, with AI agents and federated data bringing new types of concerns to the table [12], [14], and [16].

6. Discussion: Architecture as a Governance Discipline

The framework is a new paradigm to see architecture as governance rather than solely as a design choice. The value of acceleration, cost reductions, and alignment with platform upgrades is in its default configuration, but this isn't always safe. In declarative work, as in custom code, feature creep and other admin "quick fixes" and multiple sense-checkers build up as quickly. Obviously, good programs have been in operation for some time and conduct what one might describe as "code" reviews: Design review, peer review, source control, automated regression testing, and post-release monitoring are all equally applicable to declarative changes as to code. This discipline is expressed through the language of the Well-Architected pillars: make the solution more Trusted (secure, compliant, reliable), more Easy (intentional, automated, engaging), and more Adaptable (resilient, composable) [4].

There is a configuration/customization line, and its configuration is being changed in two modern ways. There's clearly something that's happening in a very more agentic architecture and an agentic architecture. There is a trend toward an agentic AI-powered architecture and an agentic architecture, for sure. Agentforce offers a novel design space for expressing capabilities and for declaratively configuring topics and actions to create action-based agents that can orchestrate flows and integrate with external services. Along with data governance [12], agent governance is a first-class issue in the Enterprise Agentic Architecture guidance for the following archetypes of single-org, multi-org, and multi-vendor orchestration. The second one is enterprise data being federated with the enterprise's external data lakehouses and unified in the Data Cloud to ease the workload of building bespoke enterprise data integrations on the core enterprise org [13] level. Architects who work in an open-ended, continuously developing profession will adapt their decision-making, approaches, and frameworks in line with their increased ability.

7. Limitations and Challenges

There are several limitations that restrict how this can be used mechanically. The number one practical hurdle is the lack of alignment: business sponsors want to preserve legacy work practices in the face of customization and ARB risk perception, as the groups can be perceived as clouding the way or being too slow. Governance has credibility with those outside of IT when translated into business terms: total cost of ownership, risk, and time to value. The same difficulty arises for non-tangible customization values, such as business case templates that account for support costs, testing burden, and platform-upgrade impact on content.

In the case of global programs, there are extra challenges. Interactions between data residency, language, currency, M&A, and regionality, and with the boundary between central and federated decision rights. As a default, it is the federated governance model recommended in Section IV, but it requires some organizational maturity, which not every program will have today, and trust between the central CoE and the regional teams, clear decision rights, and a working architecture review board. Lastly, the Salesforce platform is dynamic, and what seemed like a worthwhile custom effort a few releases ago may become a native capability in another release or two.

Table 1. *Summary of related work and gap addressed.*

Source	Focus	Key Finding	Gap Addressed
[1] Brown et al.	Technical debt	Debt arises in code and design choices	Needs Salesforce platform context
[3] Alamin et al.	Low-code platforms	Customization and integration drive most issues	Not Salesforce enterprise-specific
[4] Salesforce Architects	Well-Architected Framework	Trusted, Easy, Adaptable pillars	Not yet operationalized as a decision pipeline
[5], [6] Decision Guides	Flow vs. Apex, forms, integration	Automation density and pattern fit drive choice	Not synthesized across variables
[11] Forsgren & Humble	Continuous delivery	DevOps practices improve safety and speed	Not linked to declarative vs. code decisions
[12] Enterprise Agentic Arch.	Multi-agent and inter-org orchestration	Governance must extend to agents	Not integrated with classic config/custom framing

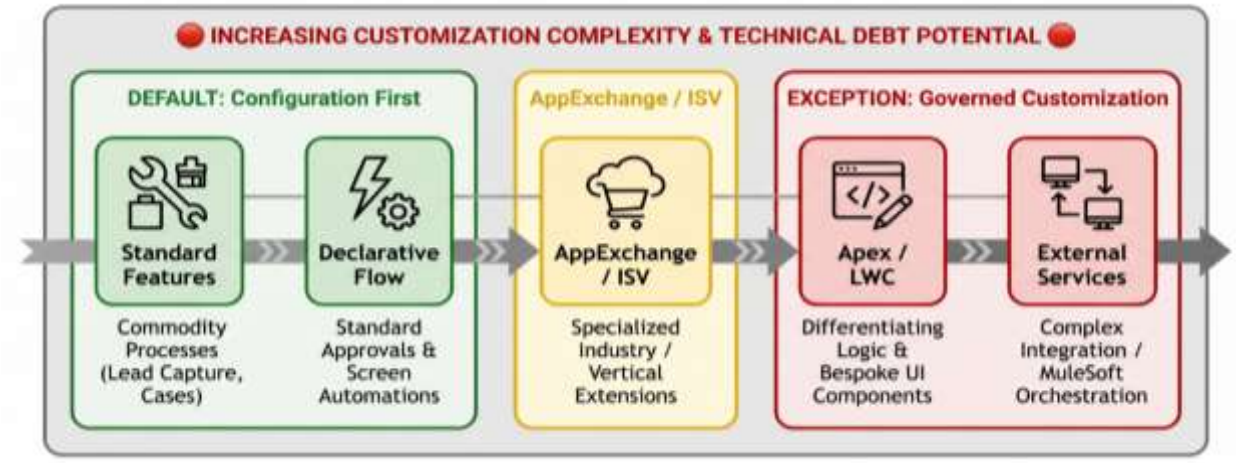

Figure 1. *Customization hierarchy illustrating the progression from standard configuration-based implementations to advanced governed customizations, highlighting the corresponding increase in technical debt and implementation complexity across standard features, declarative flows, AppExchange/ISV solutions, Apex/LWC development, and external service integrations.*

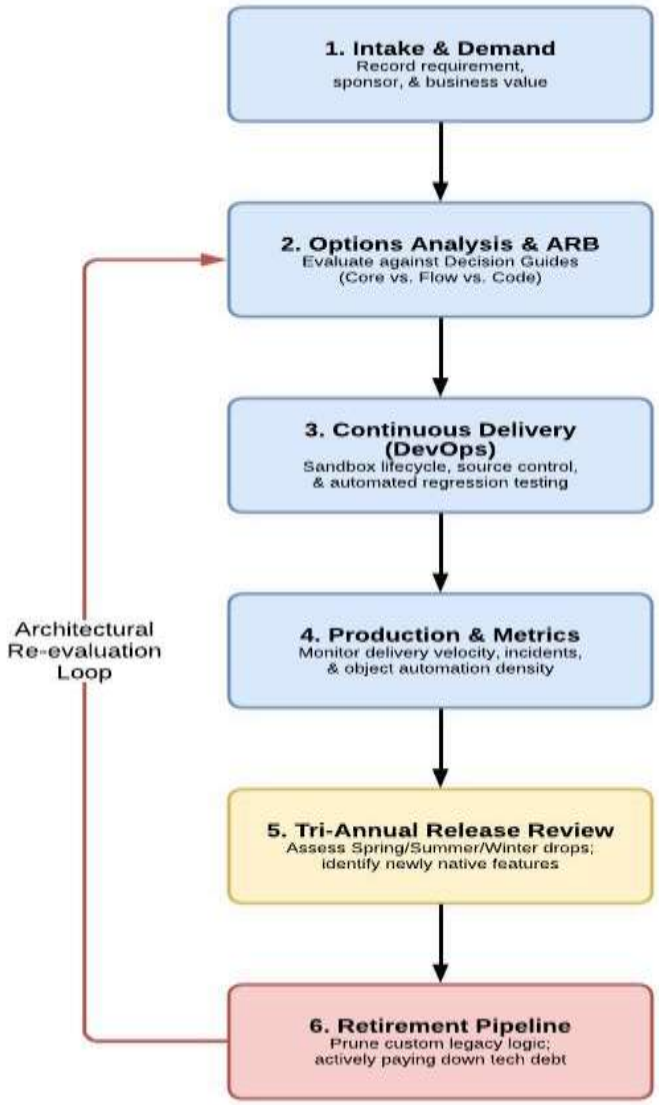

Figure 2. *Governance-driven continuous delivery lifecycle illustrating the process from requirement intake and architectural review to DevOps-enabled deployment, production monitoring, release evaluation, and retirement pipeline management for reducing technical debt and improving platform sustainability.*

Table 2. *Decision variables mapped to evidence and recommended response.*

Variable	Empirical Signal	Framework Response
Process uniqueness	Standard CRM capabilities deliver measurable productivity gains [7], [8]	Configure commodity; customize only by differentiating processes
Time-to-value	Declarative rollout is consistently faster and cheaper	Phased delivery, MVP-first, native functionality first
Integration complexity	Customization-heavy environments show more defects [3], [9]	Externalize orchestration to MuleSoft; use Data Cloud for unification
Regulatory and security	Misconfiguration is a leading risk in cloud platforms [10]	Prefer native security; custom only where native is insufficient
Maintainability	Custom-heavy estates increase testing and support costs	Track debt; review and retire customizations each release

7. Conclusions

The decision to configure your Salesforce system versus building a custom implementation of its tools is as much an architectural methodology as it is a technical one, and it shouldn't be a strictly binary choice. A configuration-first perspective is a good default for most requirements, as it enables quick delivery, lowers costs, eases upgrades, and reduces the need for rare specialist expertise. While native development is still needed in many cases where processes need to be differentiated and complex integration or regulatory requirements aren't met, it must be based on a business case and have a clearly accountable development team, with the same engineering efforts applied to it as when producing software. Doing so results in a program that is clearly more agile, less dependent on the same code, and better able to accommodate the platform's ongoing changes, grounded by the Salesforce Well-Architected Framework, guided by published Architect Decision Guides, and operating with a clear intake-to-retirement pipeline. It is not the amount of configured or coded; it's how decisions are made, recorded, and reviewed.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.
- **Use of AI Tools:** The author(s) declare that no generative AI or AI-assisted technologies were used in the writing process of this manuscript.

References

[1] N. Brown et al., "Managing technical debt in software-reliant systems," in Proc. FSE/SDP Workshop on Future of Software Engineering Research, 2010, pp. 47-52.

[2] C. Seaman and Y. Guo, "Measuring and monitoring technical debt," in Advances in Computers, vol. 82, Elsevier, 2011, pp. 25–46.

[3] M. A. A. Alamin, S. Malakar, G. Uddin, S. Afroz, T. B. Haider, and A. Iqbal, "An empirical study of developer discussions on low-code software development challenges," in IEEE/ACM MSR, 2021, pp. 46–57.

[4] Salesforce Architects, "Salesforce Well-Architected Framework: Trusted, Easy, Adaptable." [Online]. Available: https://architect.salesforce.com/well-architected

[5] Salesforce Architects, "Architect Decision Guides." [Online]. Available: https://architect.salesforce.com/decision-guides

[6] Salesforce Architects, "Record-Triggered Automation Decision Guide," 2026. [Online]. Available: https://architect.salesforce.com/docs/architect/decision-guides/guide/record-triggered.html

[7] M. Marolt, A. Pucihar, and H. D. Zimmermann, "Social CRM adoption and its impact on performance outcomes: A literature review," Organizacija, vol. 48, no. 4, pp. 260–271, 2015.

[8] U. S. Nwabekee, O. Y. Abdul-Azeez, E. E. Agu, and T. I. Ijomah, "Digital transformation in marketing

strategies: The role of data analytics and CRM tools," Int. J. Frontline Res. Sci. Technol., vol. 3, no. 2, pp. 55–72, 2024.

[9] A. C. Bock and U. Frank, "Low-code platform," Business & Information Systems Engineering, vol. 63, no. 6, pp. 733–740, 2021.

[10] Y. Zhang, H. He, O. Legunsen, S. Li, W. Dong, and T. Xu, "An evolutionary study of configuration design and implementation in cloud systems," in IEEE/ACM ICSE, 2021, pp. 188–200.

[11] N. Forsgren and J. Humble, "The role of continuous delivery in IT and organizational performance," in Proc. Western Decision Sciences Institute, 2016.

[12] Salesforce Architects, "Enterprise Agentic Architecture and Design Patterns." [Online]. Available: https://architect.salesforce.com/fundamentals/enterprise-agentic-architecture

[13] Salesforce Architects, "Data 360 Architecture." [Online]. Available: https://architect.salesforce.com/fundamentals/data-360-architecture

[14] Salesforce Engineering, "Behind the Zero-Trust Infrastructure Powering the Agentforce 360 Platform," 2025. [Online]. Available: https://engineering.salesforce.com/

[15] Salesforce Developers Blog, "Enterprise Architecture: Single-org vs. Multi-org Strategy." [Online]. Available: https://developer.salesforce.com/blogs/developer-relations/2014/10/enterprise-architecture-multi-org-strategy

[16] Salesforce Engineering, "Engineering Agentforce: Building a Modular, Multi-Model Framework for Enterprise AI Agents," 2025. [Online]. Available: https://engineering.salesforce.com/